Computing Room Acoustics Using 3D FDTD: A Cuda Approach.

COMPUTING ROOM ACOUSTICS WITH CUDA - 3D FDTD SCHEMES WITH BOUNDARY LOSSES AND VISCOSITY

Craig J. Webb and Stefan Bilbao

Acoustics and Fluid Dynamics Group/Music, University of Edinburgh

ABSTRACT

In seeking to model realistic room acoustics, direct numerical simulation can be employed. This paper presents 3D Finite Difference Time Domain schemes that incorporate losses at boundaries and due to the viscosity of air. These models operate within a virtual room designed on a detailed floor plan. The schemes are computed at 44.1kHz, using large-scale data sets containing up to 100 million points each. A performance comparison is made between serial computation in C, and parallel computation using CUDA on GPUs, showing up to 80 times speed-ups. Testing on two different Nvidia Tesla cards shows the benefits of the latest FERMI architecture for double precision floating-point computation.

Index Terms— 3D FDTD, Room Acoustics, CUDA

1. INTRODUCTION

The objective of virtual room acoustics is the realistic emulation of sound wave propagation in three-dimensional space. There are currently two approaches. In ray-based modeling, propagation is approximated to that of light and calculated to give specular and diffuse reflections. Image-source and beam-tracing [1] methods are widely used, and can be accelerated with graphics processing techniques as used in the rendering of animation [2]. However, this approach suffers from the lack of inherent diffraction properties which are essential in modeling low to mid-frequency behaviour.

The second approach is to use direct numerical calculation of the 3D wave equation. This has been demonstrated with the digital waveguide mesh [3], and methods such as adaptive rectangular decomposition [4]. FDTD schemes are a simple alternative [5], and are more efficient than the waveguide mesh. Such schemes are capable of capturing high levels of detail, but at a large (but unavoidable, due to physical considerations) computational expense. On the other hand, they are well-suited to parallel architectures such as graphics processing units (GPUs), see [6] for an overview of such techniques. Acceleration of computation using GPUs has been shown to achieve real-time simulation for small scale rooms up to audio rates of 7kHz [7]. In this paper, two varieties of FDTD scheme were tested for performance in both C and CUDA, at an audio rate of 44.1kHz. Firstly, a basic scheme using boundary losses, and then a more advanced scheme incorporating losses due to the viscosity of air.

2. FINITE DIFFERENCE SCHEMES

The starting point for acoustical FDTD simulations is the second order 3D wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u + c\alpha \nabla^2 \frac{\partial u}{\partial t} \tag{1}$$

Here, $u(x, y, z, t)$ is the target acoustical field quantity (a pressure, or possibly a velocity potential), c is the wave speed in air, ∇^2 is the 3D Laplacian, and the term with coefficient $c\alpha$ results directly from viscous damping effects, through linearization of the Navier Stokes equations [8]. α is defined as

$$\alpha \approxeq \left(1.6(\gamma - 1) + 4/3 + \frac{\eta}{\mu}\right) l/\sqrt{\gamma} \tag{2}$$

where l is the mean free path of molecules in air, γ is the ratio of specific heats, and η and μ are coefficients of viscosity. This term, when α is small, leads to frequency-dependent damping; travelling wave solutions, of frequency ω and wavenumber $\mathbf{k}$, with $|\mathbf{k}| \approxeq \omega/c$ are of the form

$$e^{-\alpha\omega^2 t/c} e^{j\omega t - \mathbf{k}\cdot\mathbf{x}} \tag{3}$$

In this paper, boundary conditions are chosen to be of simple resistive type, i.e.,

$$\frac{\partial u}{\partial t} = c\beta \mathbf{n} \cdot \nabla u \tag{4}$$

where $\mathbf{n}$ is a unit normal to a wall or obstacle, and where β is an absorption coefficient.

Simple finite difference schemes for the wave equation are described in various references; here, when the viscous damping term is added, the simplest explicit scheme will be of the form

$$\begin{aligned} u^{n+1}_{l,m,p} &= (2 + (c^2T^2 + c\alpha T)\nabla_d^2) u^n_{l,m,p} \\ &\quad -(1 + c\alpha T \nabla_d^2) u^{n-1}_{l,m,p} \end{aligned} \tag{5}$$

where T is the time step, and $u^n_{l,m,p}$ is an approximation to the continuous function $u(x, y, z, t)$, at times $t = nT$, and

locations $x = lX, y = mX$ and $z = pX$, for integer l, m, p and n, and for a grid spacing X. The seven-point discrete Laplacian ∇_d^2 is defined as

$$X^2\nabla_d^2 u_{l,m,p} = u_{l+1,m,p} + u_{l-1,m,p} + u_{l,m+1,p} + u_{l,m-1,p} + u_{l,m,p+1} + u_{l,m,p-1} - 6u_{l,m,p} \quad (6)$$

A stability condition for the scheme, over the problem interior, follows from von Neumann analysis [9], and, for a given choice of time step T (normally, the inverse of the desired audio sample rate), the grid spacing must satisfy

$$X \geq \sqrt{3c^2T^2 + 6\alpha cT} \quad (7)$$

which differs very slightly from the bound for a lossless scheme [10]. For good numerical behaviour (i.e., the least numerical dispersion), X should be chosen as close to this bound as possible.

A *basic scheme*, neglecting effects of viscosity, is obtained when $\alpha = 0$.

3. IMPLEMENTATION OF BASIC SCHEME

The basic implementation calculates propagation with losses at the boundaries using a single reflection coefficient, given by:

$$u_{l,m,p}^{n+1} = \frac{1}{1+\lambda\beta}((2 - K\lambda^2)u_{l,m,p}^n + \lambda^2 S_{l,m,p}^n - (1 - \lambda\beta)u_{l,m,p}^{n-1}) \quad (8)$$

where K is 6 in free space, 5 at a face, 4 at an edge and 3 at a corner, $\lambda = \frac{cT}{X}$, β the coefficient for losses due to boundary reflections, and S is $u_{l+1,m,p}^n + u_{l-1,m,p}^n + u_{l,m+1,p}^n + u_{l,m-1,p}^n + u_{l,m,p+1}^n + u_{l,m,p-1}^n$.

Initial prototyping was performed in Matlab to obtain a standard for correctness testing. This was then ported to C and finally CUDA to make a comparison of computation times. To go beyond a simple empty room space, a floor plan map is used to define boundaries in 2D down to the level of the grid spacing X (13.5 mm at 44.1 kHz). These boundaries are then applied at each level of the height dimension. The map defines a boundary point as a zero, and also stores the value of K in all other points (Figure 1.). By pre-defining this grid,

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	0		
0	5	6	6	6	6	6	6	6	6	6	6	6	6	6	5	0	0	0		
0	4	5	5	5	5	5	6	6	6	6	6	6	6	6	5	5	4	0		
0	0	0	0	0	0	0	5	6	6	6	5	5	5	5	6	6	6	5	0	
0	4	5	5	5	5	5	6	6	6	5	0	0	0	0	4	6	6	5	0	
0	4	5	5	5	5	5	5	5	5	4	0	0	0	0	0	4	5	4	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Fig. 1. Example of floor plan map for 20x8 grid.

2D interiors can be computed with only a small additional memory overhead.

3.1. C port

The basic scheme can be implemented using two arrays to store all data, as the calculated value $u_{l,m,p}^{n+1}$ may directly overwrite $u_{l,m,p}^{n-1}$. The 3D data arrays are arranged in a linear decomposition. The row-major format is used for each height layer, and these are then set end-to-end in a single contiguous array.

The kernel contains an outer loop over the time domain, and then three nested loops over the rows (R), columns (C) and height (Z) layers of the data. The entire data set at time step n is referred to as $\mathbf{u}^n$.

The kernel algorithm is as follows:

```
1.  Calculate linear position from R,C,Z
2.  Obtain K value from floor plan map
3.  IF NOT(K==0 OR Z==0 OR Z==Nz-1), then
4.  IF at floor and ceiling, set K = K-1
5.  IF K==5, set boundary loss coefficients
6.  Update the grid values in u^{n+1}, using six
neighbour values from u^n, and centre points
from u^n and u^{n-1}
```

This update includes boundary condition and loss in a single line of computation at step 6.

3.2. CUDA port

The kernel can be parallelized in CUDA as the update equation is applied independently at each point (not the case for all FDTD schemes, such as implicit schemes [11]). A single loop over the time domain is still required, but at each time step every point is updated by parallel CUDA threads. The threading model requires that the data be tiled into subsections because threads are grouped into blocks, and blocks grouped into a grid. A 2D block size of 16×16 was used, covering each height layer of the data. These layers are then placed side-by-side to form the block grid.

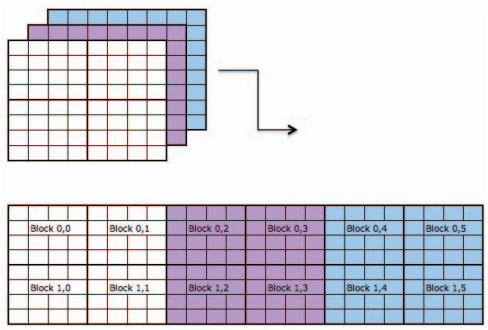

Fig. 2. Tiling of 3D data for CUDA thread block grid.

Data is then accessed by calculating the 3D data position from the thread and block IDs. From this point, the kernel is implemented as per the C port. In terms of the CUDA memory model, the data arrays are accessed directly from

global memory, and integer constants from *constant memory*. Whilst the use of *shared* memory was considered, the added complexity required to access neighbouring values in each block showed no performance benefits. The current data arrangement allows for coalesced memory access, and the latest FERMI-based GPUs provide on-chip caching [12].

A further performance consideration is the use of the three IF statements in the kernel. Whilst the first is required to prevent access to data outside of the arrays, logical arithmetic can replace those remaining. Step 4 of the kernel is written as: `IF (Z==1 OR Z==Nz-2) K = K-1`. This can be replaced with: `K = K - ( Z==1 OR Z==Nz-2)`. A similar approach can replace the conditional at step 5. However, results show that computation times increase when using this method, by a factor of 1.05.

3.3. Correctness testing

All codes were computed in double precision floating point, as previous testing showed large round-off errors at single precision. A DC-normalised audio sample was summed into the $\mathbf{u}^{n+1}$ data array at a given position at each time-step, and output taken at a given location. Although the Matlab, C and CUDA ports perform the same calculations, differences arise in finite precision. For example, Figure 3 shows the differences between computed values of the output from C and CUDA for an identical simulation. This shows relative variations in the order of 10^{-13}. Similar differences occur between Matlab and C, even when computed on the same machine. Whilst one would assume that both Matlab and C would produce the same (accurate) result, this is not the case here. Further investigation is required to establish both the causes, and effects of these differences.

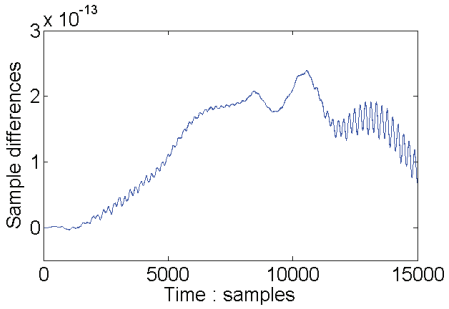

Fig. 3. Sample differences between C and CUDA outputs.

3.4. Computation times

The hardware used for performance testing was an Intel Xeon Nehalem 2.6 GHz CPU, which served a Tesla C1060 and a Tesla FERMI C2050 GPU cards. The C1060 has 240 CUDA cores, and a peak performance at double precision of 78 Gflops. The C2050 has 448 CUDA cores, and peak double precision performance of 515 Gflops. The C code ran directly on the front-end CPU, whilst the CUDA code was tested using the older Tesla and then the new FERMI card. Tests were performed at varying 3D data grid sizes, computing 1 second of output at 44.1kHz. Table 1 shows the C computation times and the speed-ups obtained for the Tesla and FERMI Tesla cards.

Grid size (total points)	C (minutes)	Tesla (speed-up)	FERMI (speed-up)
262,144	4.2	×20.7	×29.5
1,048,576	18.7	×20.6	×52.0
4,194,304	91.6	×26.1	×63.4
16,777,216	459.8	×30.5	×76.2

Table 1. Comp. times and speed-ups for basic scheme

At the 16 million point grid size, the C code takes nearly 8 hours, compared to just 6 minutes on the FERMI card.

4. IMPLEMENTATION OF ADVANCED SCHEME

The advanced scheme introduces losses due to the viscosity of air, using:

$$u^{n+1}_{l,m,p} = \frac{1}{1+\lambda\beta}((2-K\lambda^2)u^n_{l,m,p} + \lambda^2 S^n_{l,m,p} - (1-\lambda\beta)u^{n-1}_{l,m,p} + ck\alpha\nabla^2_d(u^n_{l,m,p} - u^{n-1}_{l,m,p})) \quad (9)$$

This also requires the use of three different data arrays instead of two. Computation times for the advanced scheme were:

Grid size (total points)	C (minutes)	Tesla (speed-up)	FERMI (speed-up)
262,144	6.3	×21.0	×35.6
1,048,576	29.7	×21.2	×58.4
4,194,304	146.3	×26.0	×65.2
16,777,216	761.4	×31.5	×79.8

Table 2. Comp. times and speed-ups for advanced scheme

For both the basic and advanced schemes the C codes show a ×5 increase for each ×4 increase in grid size. The CUDA codes show ×2.5 increases initially, rising to ×4. The additional memory access and computation of the advanced scheme leads to ×1.6 increase in times for both the C and CUDA codes, although the CUDA codes show smaller increases at initial grid sizes. The benefits of the FERMI architecture are clear, showing ×2.5 speed-ups over the older Tesla card, and ×80 speed-ups over C.

5. SIMULATING REALISTIC SPACES

The above testing used data grid sizes up to 16 million points. However, at 44.1kHz this only represents a space of 40m^3. Simulating larger room sizes requires far larger data grids.

5.1. Maximum room sizes

As the CUDA codes were run on single GPUs, the maximum room size is limited to the amount of global memory available on the individual card. For the FERMI Tesla this is 3Gb. Allowing for the inputs and outputs, the largest size for the three data grids was 106 million points each. This represents a room of 260m^3 (e.g., 10.6m x 8.2m x 3.0m). Computation time was 58 minutes for 1 second of output at 44.1 kHz, on the FERMI card. Various audio examples are available at : `www2.ph.ed.ac.uk/~s0956654/Site/VirtualRoomAcoustics.html`

5.2. Analysis of outputs

The inclusion of the viscosity component in the advanced scheme produces a noticeable reduction in high-frequency 'whistle'. Using a viscosity of $\alpha = 2 \times 10^{-6}$, high-frequency attenuation is in the region of 10 dB. The effect of the boundary definition using the floor plan can be seen in a plot of a single height layer, Figure 4. Boundary reflection and

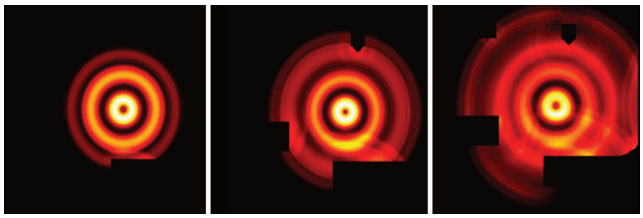

Fig. 4. 1 kHz sine wave after 78, 164, and 268 samples.

diffraction effects around objects are evident.

6. CONCLUSIONS AND FUTURE WORK

The use of 3D FDTD schemes to model room acoustics at 44.1 kHz can yield highly detailed results, at the expense of long computation times. Parallel GPU threading shows speed-ups of up to 80 times over serial computation. This is the difference between over 3 days in C, to 1 hour in CUDA, for a moderately-sized room (260 m^3). The inclusion of viscosity effects improved the quality of the reverberation for this basic model, and the detailed floor plans allow for realistic room spaces with minimal overhead.

Further development of the scheme will consider more complex boundary conditions and absorption parameters, as well as investigation of the correctness issues observed during testing. The use of multiple GPUs using MPI programming should allow for greater acceleration, or the simulation of larger spaces.

7. REFERENCES

[1] I. A. Drumm, "The application of adaptive beam tracing and managed DirectX for the visualisation of virtual environments," in *IV '05: Proc. of the Ninth Int. Conf. on Inf. Visualisation*, Washington, DC, USA, 2005, pp. 961–965, IEEE CompSoc.

[2] N. Rober, U.Kaminski, and M. Masuch, "Ray acoustics using computer graphics technology," in *Proc. of the 10th Int. Conf. on Digital Audio Effects*. DAFx-07, Bordeaux, France, September 2007.

[3] L. Savioja, M. Karjalainen, and T. Takala, "DSP formulation of a finite difference method for room acoustics," in *Proc. of Nordic Signal Processing Symp.* NORSIG, 1996, pp. 455–458.

[4] N. Raghuvanshi, R. Narain, and M. Lin, "Efficient and accurate sound propagation using adaptive rectangular decomposition," in *IEEE Trans. on Visualisation and computer graphics*, 2009, vol. 15, pp. 789–801.

[5] A. Southern, D. Murphy, and J. Wells, "Rendering walkthrough auralisations using wave-based acoustical models," in *Proc. of the 17th European Signal Processing Conf.* EUSIPCO 09, Glasgow, UK, 2009.

[6] L. Savioja, D. Manocha, and M. Lin, "Use of GPUs in room acoustic modeling and auralization," in *Proc. Int. Symp. on Room Acoustics*. ISRA, Melbourne, 2010.

[7] L. Savioja, "Real-time 3D finite-difference time-domain simulations of low and mid-frequency room acoustics," in *13th Int. Conf on Digital Audio Effects*, Sept, 2010.

[8] P. Morse and U. Ingard, *Theoretical Acoustics*, Princeton University Press, Princeton, New Jersey, 1968.

[9] J. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Wadsworth and Brooks/Cole Advanced Books and Software, Pacific Grove, California, 1989.

[10] S. Bilbao, *Wave and Scattering Methods for Numerical Simulation*, John Wiley and Sons, Chichester, UK, 2004.

[11] K. Kowalczyk and M. van Walstijn, "A comparison of nonstaggered compact FDTD schemes for the 3D wave equation," in *IEEE Int. Conf. on Acoustics Speech and Signal Processing*. ICASSP, Mar 2010, pp. 197–200.

[12] Nvidia Corp, "Fermi compute architecture whitepaper," http://developer.nvidia.com/object/gpucomputing.html, Oct, 2010.